CMSC424: Database Design
SQL

February 19, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Plan

- Project 1 discussion
- Entity-Relationship Model Details
- Anatomy of a Web Application
  - Project 2
- Converting from E/R Model to Relational Schema
Today’s Plan

- Project 1 discussion
- Entity-Relationship Model Details
- Anatomy of a Web Application
  - Project 2
- Converting from E/R Model to Relational Schema
Entity-Relationship Model

- Two key concepts
  - **Entities**: An object that *exists* and is *distinguishable* from other objects
    - Examples: Bob Smith, BofA, CMSC424
  - Have **attributes** (people have names and addresses)
  - Form **entity sets** with other entities of the same type that share the same properties
    - Set of all people, set of all classes
  - Entity sets may overlap
    - Customers and Employees
Two key concepts

- **Relationships**: Relate 2 or more entities
  - E.g. Bob Smith *has account at* College Park Branch
  - Form *relationship sets* with other relationships of the same type that share the same properties
    - Customers *have accounts at* Branches
  - Can have attributes:
    - *has account at* may have an attribute *start-date*
  - Can involve more than 2 entities
    - Employee *works at* Branch *at* Job
Entities and relationships

Two Entity Sets

instructor

student

Advisor Relationship, with and without attributes

instructor

student
ER Diagram

Alternative representation, used in the book in the past

Both notations used commonly
Rest of the class

- Details of the ER Model
  - How to represent various types of constraints/semantic information etc.

- Design issues

- A detailed example
Next: Relationship Cardinalities

- We may know:
  - One customer can only open one account
  - OR
  - One customer can open multiple accounts

- Representing this is important

- Why?
  - Better manipulation of data
    - If former, can store the account info in the customer table
  - Can enforce such a constraint
    - Application logic will have to do it; NOT GOOD
  - Remember: If not represented in conceptual model, the domain knowledge may be lost
Express the number of entities to which another entity can be associated via a relationship set

Most useful in describing binary relationship sets
Mapping Cardinalities

- **One-to-One**
  - Instructor advises Student

- **One-to-Many**
  - Instructor advises Student

- **Many-to-One**
  - Instructor advises Student

- **Many-to-Many**
  - Instructor advises Student
Express the number of entities to which another entity can be associated via a relationship set

Most useful in describing binary relationship sets

N-ary relationships?
- More complicated
- Details in the book
Next: Types of Attributes

- Simple vs Composite
  - Single value per attribute?

- Single-valued vs Multi-valued
  - E.g. Phone numbers are multi-valued

- Derived
  - If date-of-birth is present, age can be derived
  - Can help in avoiding redundancy, enforcing constraints etc...
Types of Attributes

- **Primary key underlined**: ID
- **Composite**: name, first_name, middle_initial, last_name, address, street, street_number, street_name, apt_number, city, state, zip
- **Multi-valued**: { phone_number }
- **Derived**: date_of_birth, age ()
What attributes are needed to represent a relationship completely and uniquely?

- Union of primary keys of the entities involved, and relationship attributes

\{instructor.ID, date, student.ID\} describes a relationship completely
Is \{\text{student\_id}, \text{date}, \text{instructor\_id}\} a candidate key?

- No. Attribute \text{date} can be removed from this set without losing key-ness.
- In fact, union of primary keys of associated entities is always a superkey.

\begin{figure}[h]
\centering
\includegraphics[width=0.6\textwidth]{er_diagram.png}
\caption{E-R diagram with an attribute attached to a relationship set.}
\end{figure}
Is \{\text{student\_id}, \text{instructor\_id}\} a candidate key?

- Depends

![Relationship Set Diagram](image)

**Figure 7.8** E-R diagram with an attribute attached to a relationship set.
Is \{\text{student}\_id, \text{instructor}\_id\} a candidate key?

- Depends

If one-to-one relationship, either \{\text{instructor}\_id\} or \{\text{student}\_id\} sufficient

- Since a given \text{instructor} can only have one \text{advisee}, an instructor entity can only participate in one relationship

- Ditto \text{student}
Is \{\text{student}_\text{id}, \text{instructor}_\text{id}\} a candidate key?

- Depends

If one-to-many relationship (as shown), \{\text{student}_\text{id}\} is a candidate key

- A given instructor can have many advisees, but at most one advisor per student allowed
Relationship Set Keys

- General rule for binary relationships
  - one-to-one: primary key of either entity set
  - one-to-many: primary key of the entity set on the many side
  - many-to-many: union of primary keys of the associate entity sets

- n-ary relationships
  - More complicated rules
What have we been doing

Why?

Understanding this is important
  - Rest are details !!
  - That’s what books/manuals are for.
Sometimes a relationship associates an entity set to itself

- Need “roles” to distinguish
Weak Entity Sets

- An entity set without enough attributes to have a primary key
- E.g. Section Entity
- Still need to be able to distinguish between weak entities
  - Called “discriminator attributes”: dashed underline

![Diagram showing entity relationships between course, sec_course, and section entities]
Sometimes a relationship associates an entity set to itself

Need “roles” to distinguish
Similar to object-oriented programming: allows inheritance etc.
Aggregation

- No relationships allowed between relationships
- Suppose we want to record evaluations of a student by a guide on a project
Thoughts...

- Nothing about actual data
  - How is it stored?

- No talk about the query languages
  - How do we access the data?

- Semantic vs Syntactic Data Models
  - Remember: E/R Model is used for conceptual modeling
  - Many conceptual models have the same properties

- They are much more about representing the knowledge than about database storage/querying
Basic design principles

- Faithful
  - Must make sense
- Satisfies the application requirements
- Models the requisite domain knowledge
  - If not modeled, lost afterwards
- Avoid redundancy
  - Potential for inconsistencies
- Go for simplicity

Typically an iterative process that goes back and forth
Design Issues

- Entity sets vs attributes
  - Depends on the semantics of the application
  - Consider *telephone*

- Entity sets vs Relationship sets
  - Consider *loan*

- N-ary vs binary relationships
  - Possible to avoid n-ary relationships, but there are some cases where it is advantageous to use them

- It is not an exact science !!
Recap

- Entity-relationship Model
  - Intuitive diagram-based representation of domain knowledge, data properties etc...
  - Two key concepts:
    - Entities
    - Relationships
  - We also looked at:
    - Relationship cardinalities
    - Keys
    - Weak entity sets
    - ...
Recap

- Entity-relationship Model
  - No standardized model (as far as I know)
    - You will see different types of symbols/constructs
  - Easy to reason about/understand/construct
  - Not as easy to implement
    - Came after the relational model, so no real implementation was ever done
    - Mainly used in the design phase
Today’s Plan

- Project 1 discussion
- Entity-Relationship Model Details
- Anatomy of a Web Application
  - Project 2
- Converting from E/R Model to Relational Schema
Application Architecture Evolution

- Three distinct eras of application architecture
  - Mainframe (1960’s and 70’s)
  - Personal computer era (1980’s)
  - Web era (mid 1990’s onwards)
  - Web and Smartphone era (2010 onwards)
Web or Mobile Applications

- Web browsers and mobile applications have become de facto standard user interface
  - Wide cross-platform accessibility
  - No need to download something

Three Tier or Two Tier Architectures
What runs where?

1. Web Browser (Firefox, Chrome, Safari, Edge)
2. HTML to render webpages
3. Javascript for “client-side scripting” (running code in your browser without contacting the server)
4. Flash (not supported much – too much security risk)
5. Java “applets” – less common today

- Flask, Django, Tomcat, Node.js, and others
- Accept requests from the client and pass to the application server
- Pass application server response back to the client
- Support HTTP and HTTPS connections

- Encapsulates business logic
- Needs to support different user flows
- Needs to handle all of the rendering and visualization
- Ruby-on-rails, Django, Flask, Angular, React, PHP, and many others

- PostgreSQL, Oracle, SQL Server, Amazon RDS (Relational Databases)
- MongoDB (Document/JSON databases)
- SQLite --- not typically for production environments
- Pretty much any database can be used…
Application Server

- Fair amount of complexity in here
- Need to deal with “user flows”
  - Different types of actions user can take
  - Typically multi-step flows across screens
  - What happens when a user clicks this vs that
- Need to interface with the database
  - To look up the information needed to show to a user
  - To save updates made by the user
- Need to deal with rendering of the information
  - Generating the HTML to show the information to the user
  - Handling the “forms” for when a user makes changes
**Django Architecture**

"urls.py":
- Map incoming URLs to Views

"views.py":
- Fetch data from DB through models
- Do computations, create new objects, etc.
- Send data to "template"

"models.py":
- Define the different types of entities and relationships
- Think E/R Model more than Relational
- Entities map to classes – relationships may be implicit or explicit
- Django takes care of creating the RDBMS schema

Templates directory:
- Create dynamic HTML using the data sent by "view.py"
- A mix of HTML and embedded Django code

Figure from: https://www.researchgate.net/figure/Specific-Django-architecture_fig1_332023947
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.mainindex, name='mainindex'),

    url(r'^user/(?P<user_id>[0-9]+)/$', views.userindex, name='userindex'),

    url(r'^event/(?P<event_id>[0-9]+)/$', views.eventindex, name='eventindex'),

    url(r'^calendar/(?P<calendar_id>[0-9]+)/$', views.calendarindex, name='calendarindex'),

    url(r'^user/(?P<user_id>[0-9]+)/createevent$', views.createevent, name='createevent'),

    url(r'^user/(?P<user_id>[0-9]+)/submitcreateevent/$', views.submitcreateevent, name='submitcreateevent'),

    url(r'^user/(?P<user_id>[0-9]+)/createdevent/(?P<event_id>[0-9]+)/$', views.createdevent, name='createdevent'),

    url(r'^waiting/user/(?P<user_id>[0-9]+)/calendar/(?P<calendar_id>[0-9]+)/$', views.waiting, name='waiting'),

    url(r'^summary$', views.summary, name='summary'),
]
def eventindex(request, event_id):
    event = Event.objects.get(pk=event_id)
    statuses = [(c.title, BelongsTo.Status(BelongsTo.objects.get(event=event, calendar=c).status)) for c in event.calendars]
    context = {'event': event, 'statuses': statuses}
    return render(request, 'mycalendar/eventindex.html', context)

- Get the event object from the database
- Get all the “status” associated with it
- Create the “context” object
- Pass it to “eventindex” template
Project 2: “eventindex.html”

{# if event %}
    <h3> Event Information </h3>
    <b> Event Title: </b> {{ event.title }} <br>
    <b> Start Time: </b> {{ event.start_time }} <br>
    <b> End Time: </b> {{ event.end_time }} <br>
    <h4> Invited Calendars: </h4>
    <table style="border:2px solid black; padding:10px; border-collapse: collapse">
        <tr><th style="border:2px solid black"> Calendar Name </th> <th style="border:2px solid black"> Status </th></tr>
        {% for x, y in statuses %}
            <tr><td style="border:2px solid black"> {{ x }} </td> <td style="border:2px solid black"> {{ y.label }} </td></tr>
        {% endfor %}
    </table>
{# endif %}

Django command – pulls the title from the “event” object passed by views.py

You can do for loops and conditionals, but not arbitrary python (that’s for “views.py”
class Event(models.Model):
    title = models.CharField(max_length=50)
    start_time = models.DateTimeField()
    end_time = models.DateTimeField()
    calendars = models.ManyToManyField(Calendar, through='BelongsTo')
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    def __str__(self):
        return self.title

Maps to a table in the backend (SQLite3) Database

sqlite> .schema mycalendar_event

CREATE TABLE IF NOT EXISTS "mycalendar_event"
    ( "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
      "start_time" datetime NOT NULL,
      "end_time" datetime NOT NULL,
      "created_by_id" integer NOT NULL REFERENCES "mycalendar_user" ("id")
      DEFERRABLE INITIALLY DEFERRED,
      "title" varchar(50) NOT NULL);